CS103L SPRING 2020

UNIT 2: CONTROL STRUCTURES. IF... ELSE, FOR, WHILE

TEXT

LEARNING OBJECTIVES

- Understand control flow statements:
  - if, else-if, else
  - while, do-while
  - for

TEXT

QUICK WARMUP

- Write a program to ask the user to enter two integers representing hours then minutes
- In class exercises
  - http://bytes.usc.edu/cs103/in-class-exercises/in-class-exercises-set1/
- printseconds
- Skeleton code including variables is already started

TEXT

COMPARISONS AND LOGICAL OPERATORS

- Often we want to check if a *condition* is true or false
  - Is x == 100? Is y < 35.5? Is x > z?
- Control flow statements require conditions
- In C/C++ the integer value 0 means ‘false’. Any value other than 0 is ‘true’
  - Have bool type and true/false constants to help
  - Behind the scenes ‘true’ == 1, ‘false’ == 0

CONDITIONS

- Conditions usually arrive from comparisons and logical operators
- Comparisons:
  - ==, !=, >, <, >=, <=
- Equals to, not equals, greater than, less than, greater than or equal, less than or equal

LOGICAL OPERATORS

- Combine the result of expressions ‘logically’
- Logical AND: && operator expr_1 && expr_2
  - x == 1 && y <= 2
- Logical OR: || operator expr_1 || expr_2
  - x >= 5 || x < 0
- Logical NOT: ! operator !expr_1
  - !(x >= 5 || x < 0)
  - !x && y < 0
- Operator precedence: ! then && then ||

A	B	AND
False	False	False
False	True	False
True	False	False
True	True	True

A	B	OR
False	False	False
False	True	True
True	False	True
True	True	True

A	NOT
False	True
True	False

TEXT

LOGICAL OPERATOR PRACTICE

- x=100;y=-3;z=0
- !x || y && !z
- With parens: ((!x) || (y && (!z)))

TEXT

LOGICAL OPERATOR PRACTICE

- Which of the following does *not* check if an integer x is in the range [-1 to 5] (inclusive)
  - x >= -1 && x <=5
  - -1 <= x <= 5
  - !(x < -1 || x > 5)
  - x > -2 && x < 6

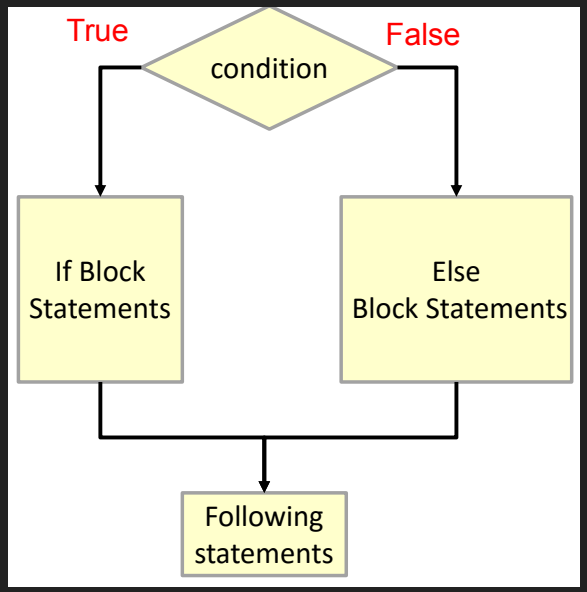
TEXT

IF STATEMENTS

- Controlling the flow of your code

CONTROL STATEMENT #1: IF . . . ELSE IF . . . ELSE

- Used to control what code executes based on one or more conditions
- Form 1: basic if statement
- else block is optional



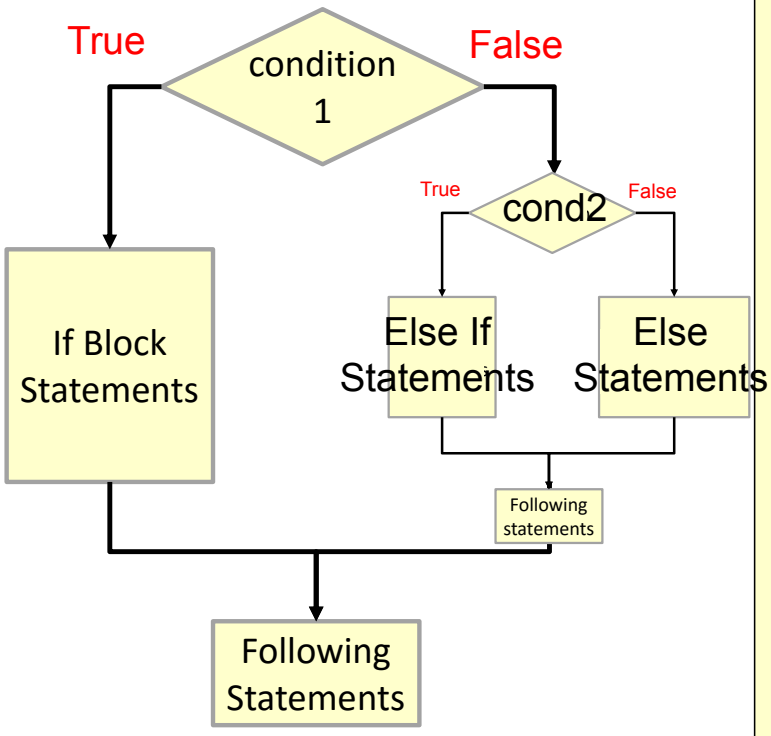

```
if (condition1)
{
  // executed if condition1 is True
}
else
{
  // executed if neither condition
  // above is True
}

// following statements
```

IF STATEMENT

- Use if to execute only parts of your code
- else if is optional, can have as many as you need
- final else is also optional, will execute if *none* of the conditions are true
- Use {} brackets to associate code with the condition

```
if (condition1)
{
  // executed if condition1 is True
}
else if (condition2)
{
  // executed if condition2 is True
  //  but condition1 was False
}
else if (condition3)
{
  // executed if condition3 is True
  //  but condition1 and condition2
  //  were False
}
else
{
  // executed if neither condition
  // above is True
}
```

FULL IF STATEMENT FLOW CHART




```
if (condition1)
{
  // executed if condition1 is True
}
else if (condition2)
{
  // executed if condition2 is True
  //  but condition1 was False
}
else
{
  // executed if neither condition
  // above is True
}
// following statements
```

TEXT

IN CLASS EXERCISES

- discount
- weekday
- N-th

TEXT

STYLE NOTES

- Code “style”
  - What do we mean by that?
- Often there are many, many different ways to write even simple code
- Try to choose version that is clear, shows intent

TEXT

STYLE QUESTION

- Which is better?
  - Why?

```
int x;
cin >> x;

if( x >= 0) {
    cout << “Positive”;
}
if( x < 0 ) {
    cout << “Negative”;
}
```

```
int x;
cin >> x;

if( x >= 0) {
    cout << “Positive”;
}
else {
    cout << “Negative”;
}
```

TEXT

COMMON BUG

- What's wrong here?

```
int x;
cin >> x;

if( x = 1) {
    cout << “x is 1!” << endl;
}
else {
    cout << “x is not 1.” << endl;
}
```

TEXT

= VS ==

- Common mistake to use = (assignment) instead of == (equals to)
- The = operator returns the value assigned, so (x = 1) returns 1
- if(x = 1) will always get 1 → true
  - if() block will always execute!

```
int x;
cin >> x;

if( x = 1) {
    cout << “x is 1!” << endl;
}
else {
    cout << “x is not 1.” << endl;
}
```

= VS == FIXED

- Fixed code

```
int x;
cin >> x;

if( x == 1) {
    cout << “x is 1!” << endl;
}
else {
    cout << “x is not 1.” << endl;
}
```

? OPERATOR

- Often we find ourselves writing code of the form:

```
if (x>0)
{
   z = 2;
}
else
{
   z = 1;
}
```

- ? is a short cut: z = x > 0 ? 2 : 1;
- Syntax: *condition ? expr_if_true : expr_if_false;*

LOOPS

- Doing something more than once

WHY DO WE NEED LOOPS?

- Almost *all* programs have at least one loop…
  - Print out numbers 1 - 100
  - Play a game of rounds until there is a winner
  - Can we do this without loops?
    - Maybe?

```
#include <iostream>
using namespace std;

int main()
{
  cout << 1 << endl;
  cout << 2 << endl;
  ...
  cout << 100 << endl;
  return 0;
}
```

```
#include <iostream>
using namespace std;

int main()
{
  bool gameOver;
  gameOver = take_turn();
  if( ! gameOver ){
    gameOver = take_turn();
    if( ! gameOver ) {
       ...
    {
  }
}
```

Assume this produces a true/false result indicating if the game is over after performing a turn

TEXT

LOOP TYPE #1: WHILE LOOP

- While loop
  - Do something while a condition is true
- Two forms:
  - while(condition){ //something }
  - do { //something } while(condition)

TEXT

WHILE LOOP

- *condition* is evaluated first
  - If *condition* is true, body is executed
  - After body, *condition* is checked again
- Body *should* update condition
  - Why?
- How many times will body execute?

```
// While Type 1:
while(condition)
{
  // code to be repeated
  // (should update condition)
}
```

TEXT

DO-WHILE LOOP

- Body is executed
- Then *condition* is checked, if true, body is executed again
- Body should update condition
- How many times will body execute?

```
// While Type 2:
do {
  // code to be repeated
  // (should update condition)
} while(condition);
```

WHILE LOOP = REPEATING IF

- While loop is like a repeating if statement
- If your description of the problem (algorithm) contains:
  - “until <xxx> is true”
  - “as long as <yyy> is bigger than <x>
  - “while the player’s guess is wrong”
- Then you need a while loop

```
// guessing game
bool guessedCorrect = false;
if( !guessedCorrect )
{
    guessedCorrect = guessAgain();
}
// want to repeat if cond. check again
if( !guessedCorrect )
{
    guessedCorrect = guessAgain();
} // want to repeat if cond. check again
```

An if-statement will only execute once

```
// guessing game
bool guessedCorrect = false;

while( !guessedCorrect )
{
    guessedCorrect = guessAgain();
}
```

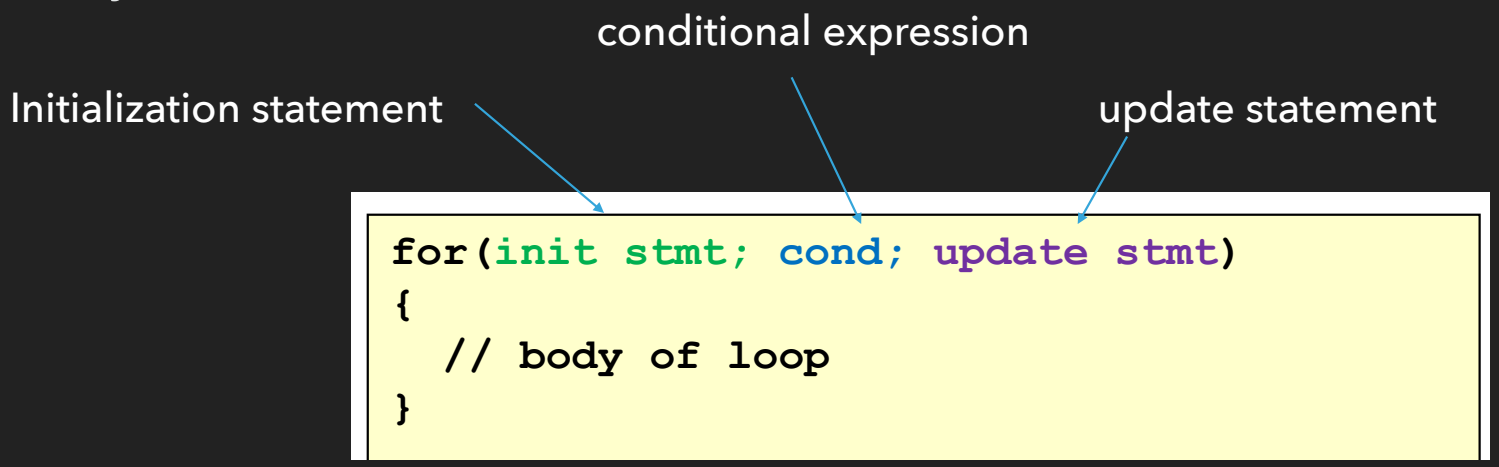
A 'while' loop acts as a repeating 'if' statement

IN CLASS EXERCISES

- countodd

FOR LOOP

- Very common looping structure
- Anatomy:



TEXT

FOR LOOP

- Initialization statement:
  - Any valid statement, executed first
- Conditional Expression:
  - Expression evaluated next
- Body:
  - Executed if *conditional expression* is true
- Update statement:
  - Any valid statement, executed after body
- Condition is checked again after Update and body is executed if true… until condition is false

```
for(init stmt; cond; update stmt)
{
  // body of loop
}
```

TYPICAL FOR LOOP

- Initialization statement
  - Used to initialize counter variable
- Condition
  - Check to see if counter is past desired range
- Update statement
  - Used to update counter variable

```
for(init stmt; cond; update stmt)
{
  // body of loop
}

// Outputs 0 1 2 3 4 (on separate lines)
for(i=0; i < 5; i++){
  cout << i << endl;
}

// Outputs 0 5 10 15 … 95 (on sep. lines)
for(i=0; i < 20; i++){
  cout << 5*i << " is a multiple of 5";
  cout << endl;
}
// Same output as previous for loop
for(i=0; i < 100; i++){
  if(i % 5 == 0){
    cout << i << " is a multiple of 5";
    cout << endl;
  }
}

// compound init and update stmts.
for(i=0, j=0; i < 20; i++,j+=5){
  cout << j << " is a multiple of 5";
  cout << endl;
}
```

TEXT

FOR VS. WHILE LOOP

- When to use while (rule of thumb)?
  - When we don't know (can't quantify) how many times the loop will run
- When to use for (rule of thumb)?
  - When we know how many times, or can quantify how many times the loop will run
- Any for loop can be turned into while loop
  - And vice-versa
  - Try it!

```
//guessing game
bool guess = false;
while(!guess)
{
    guess = guessAgain();
}
```

```
int x;
cin >> x;
for(i = 0; i < x; i++)
{
   cout << i << endl;
}
```

here we don't know x ahead of time, but we can quantify the loop count

TEXT

LOOP PRACTICE

- Write a loop to compute the Liebniz approximation to $\pi/4$ using the first 10 terms:
  - $\pi/4 = 1/1 - 1/3 + 1/5 - 1/7 + 1/9...$
- In-class exercises: liebnizapprox
- Tip: write out a table to determine the pattern

Iteration	fraction	Sign
1	1/1	+
2	1/3	-
3	1/5	+
...	...	...

TEXT

MORE LOOP PRACTICE

- Write for loops to compute the following approximations out to 10 terms
- $e^x = 1 + x + x^2/2! + x^3/3! + x^4/4!...$
- Wallis approximation for $\pi/2$
- $\pi/2 = 2/1 * 2/3 * 4/3 * 4/5 * 6/5 * 6/7 * 8/7 ...$
  - In class exercises: wallisapprox

INFINITE LOOPS

- All programmers do it from time-to-time…
- An infinite loop is a loop that never exits
  - while or for loop where condition is always true:

```
#include <iostream>
using namespace std;
int main()
{ int val;
  bool again = true;
  while(again = true){
    cout << "Enter an int or -1 to quit";
    cin >> val;
    if( val == -1 ) {
       again = false;
    }
  }
  return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
  int i=0;
  while( i < 10 ) {
     cout << i << endl;
     i + 1;
  }
  return 0;
}
```

INFINITE LOOPS…INFINITE LOOPS…INFINITE LOOPS…INFINITE LOOPS…

- If you write one, hit 'ctrl-c' on your keyboard to stop the program running
  - If you're already in the debugger, you can figure out which loop is infinite
- Fixed:

```
#include <iostream>
using namespace std;
int main()
{ int val;
  bool again = true;
  while(again == true){
    cout << "Enter an int or -1to quit";
    cin >> val;
    if( val == -1 ) {
       again = false;
    }
  }
  return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
  int i=0;
  while( i < 10 ) {
     cout << i << endl;
     i = i + 1;
  }
  return 0;
}
```

NEAT WHILE-LOOP TRICK TO READ MULTIPLE INPUTS

- What's going on here?
- Observe: if (cin >> val) is successful the expression is 'true'
- What does that do?
- Ctrl-D makes (cin >> val) fail → false
- Nice way to loop and read inputs
  - We will use this technique again later

```
#include <iostream>
using namespace std;
int main()
{ int val;
  // reads until user hits Ctrl-D
  // which is known as End-of-File(EOF)
 cout << "Enter an int or Ctrl-D ";
 cout << " to quit: " << endl;

 while(cin >> val){
    cout << "Enter an int or Ctrl-D "
    cout << " to quit" << endl;
    if(val % 2 == 1){
      cout << val << " is odd!" << endl;
    }
    else {
      cout << val << " is even!" << endl;
    }
  }
  return 0;
}
```

SIDE NOTE ON SYNTAX

- Remember { } is a compound statement.
- if, while, do-while only require a statement for the body
- So sometimes we write “single statement bodies”
- Personal preference
  - I usually always go with { }… why?
  - First three examples are OK- I don’t like the third

```
if (x == 5)
  y += 2;
else
  y -= 3;


for(i = 0; i < 5; i++)
  sum += i;


while(sum > 0)
  sum = sum/2;


for(i = 1 ; i <= 5; i++)
  if(i % 2 == 0)
     j++;
```

TEXT

MORE IN-CLASS EXERCISES

- Determine if a number is prime or not?
  - How to determine if prime (identify the algorithm)
  - What numbers could be factors?
  - When can we stop?
- Reverse digits of an integer
  - User enters 123 → 321
  - User enter -5467 → -7645

TEXT

NESTED LOOPS

- Often we want to go over all combinations of two (or more variables) over some range
- 2D coordinates for images, 3D coordinates, row-column (matrix operations)
- Usually two (or three) for loops, tightly nested

TEXT

NESTED LOOPS

- For each iteration of the outer loop, one complete iteration of the inner loop runs
- This program outputs:
  - 0 0
  - 0 1
  - 0 2
  - 1 0
  - 1 1
  - 1 2

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  for(int i=0; i < 2; i++){
    for(int j=0; j < 3; j++){
      // Do something based
      // on i and j
      cout << i << " " << j;
      cout << endl;
    }
  }
  return 0;
}
```

NESTED LOOPS

- Commonly used for row-column operations like images or matrices
- Here we're printing out row*column
- Output isn't pretty though…
  - 123456789101112246810121 4…

	1	2	3
1	1	2	3
2	2	4	6
3	3	6	9

```
#include <iostream>

using namespace std;

int main()
{
  for(int r=1; r <= 12; r++){
    for(int c=1; c <= 12; c++){
       cout << r*c;
    }
  }
  return 0;
}
```

NESTED LOOP EXAMPLE

- Fix #1: add newline at the end of each row.
- Where does that go in the code?
- Now we get:
  - 1 2 3 4 5 6 7 8 9 10 11 12
  - 2 4 6 8 10 12 14 16 18 20 22 24
  - …

```
#include <iostream>

using namespace std;

int main()
{
  for(int r=1; r <= 12; r++){
    for(int c=1; c <= 12; c++){
       cout << " " << r*c;
    }
    cout << endl;
  }
  return 0;
}
```

NESTED LOOP EXAMPLE

- cout can be “fed” things that change its’ behavior
  - setw(x): print each thing padded to a width of 4 (with spaces)
- Fixes our printing problem

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  for(int r=1; r <= 12; r++){
    for(int c=1; c <= 12; c++){
       cout << setw(4) << r*c;
    }
    cout << endl;
  }
  return 0;
}
```

TEXT

IN-CLASS EXERCISES

- 5-per line (A, B, C)

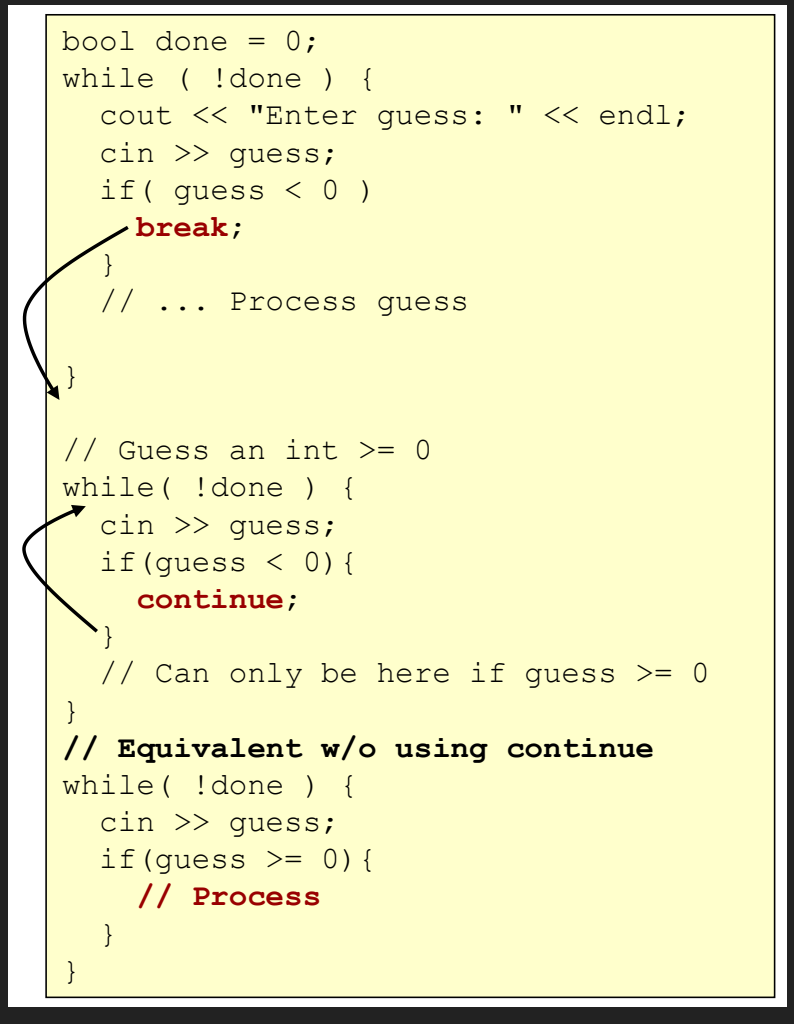
TEXT

BREAK AND CONTINUE

- Sometimes we want to exit a loop early
- Or we want to skip to the next iteration of the loop
- break
  - Jump immediately out of the loop to the code that follows
  - None of the loop conditions are checked, or update statement run (for loop)
- continue
  - Jump immediately to the top of the loop
  - Checks the condition (for, while) and runs the update statement (for loop)

BREAK AND CONTINUE

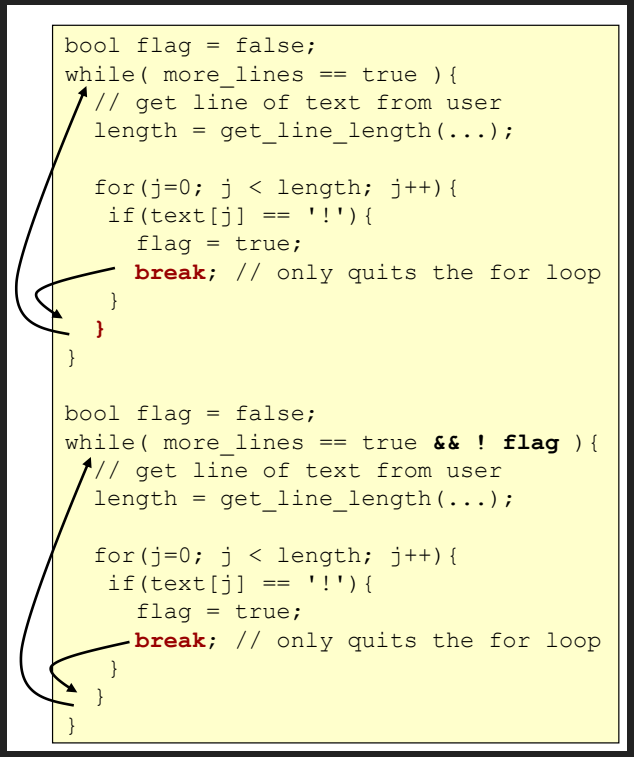
- Examples:
  - Games
  - Searching in an array or list
  - When an error occurs



```
bool done = 0;
while ( !done ) {
  cout << "Enter guess: " << endl;
  cin >> guess;
  if( guess < 0 )
    break;
  }
  // ... Process guess

}

// Guess an int >= 0
while( !done ) {
  cin >> guess;
  if(guess < 0){
    continue;
  }
  // Can only be here if guess >= 0
}
// Equivalent w/o using continue
while( !done ) {
  cin >> guess;
  if(guess >= 0){
    // Process
  }
}
```

BREAK AND CONTINUE

- Only apply to the loop they're in, not all nested for or while loops



```
bool flag = false;
while( more_lines == true ){
  // get line of text from user
  length = get_line_length(...);

  for(j=0; j < length; j++){
   if(text[j] == '!'){
     flag = true;
     break; // only quits the for loop
   }
  }
}

bool flag = false;
while( more_lines == true && ! flag ){
  // get line of text from user
  length = get_line_length(...);

  for(j=0; j < length; j++){
   if(text[j] == '!'){
     flag = true;
     break; // only quits the for loop
   }
  }
}
```

PRE-PROCESSOR DIRECTIVES

- C++ has what is known as a pre-processor
- Scans your code looking for lines that start with #
- #include literally “includes” the whole of another file in your source code
  - We use to bring in elements of C++ standard library
  - Later we’ll write our own “header” files and #include them too
- #define *pattern_1 pattern_2*
  - Looks for all examples of the text in pattern_1 and replaces with pattern_2
  - #define PI 3.141592 (replaces all occurrences of PI with 3.141592)

TEXT

GENERATING RANDOM NUMBERS

- Generating (pseudo) random numbers is very important for lots of applications in computer science
- We'll need them for PA2
- #include <cstdlib>
  - Gives us the function rand() which returns an integer between 0 and RAND_MAX (a constant also defined by <cstdlib>
  - int r = rand();
- How to generate uniform (pseudo) random numbers between [0,1]?
  - double r = ((double)rand())/RAND_MAX;

TEXT

PSEUDO-RANDOM

- Why do I keep saying pseudo random?
- Most random number generators in CS are pseudo random (PRNG)
  - They generate long sequences of random-looking numbers
  - Aren't really random, sometimes (usually for us) this is OK
- "Seeding" the PRNG gives the offset into the sequence
- If you seed the same PRNG with the same seed, you'll always get the same sequence
  - Useful for testing

TEXT

SEEDING PRNG

- If you are testing, seed with 0, or whatever # you want
- When you want the PRNG to produce “random” results, seed with “time”
  - Integer # of seconds since Jan. 1, 1970
  - #include <ctime>
- Ex:
  - srand(0);
  - srand(time(0));
- Only call srand() once at the beginning of your program. Otherwise you might get strange results.

TEXT

SRAND() AND RAND()

- Example:
  - srand(time(0));
  - int r1 = rand();
  - int r2 = rand();
- Sequence of numbers in r1 and r2 will be different each time you run your program.

TEXT

IN CLASS EXERCISES

- randcalls
- seed
- seedtime

TEXT

ACKNOWLEDGEMENTS

- Yellow code samples courtesy Mark Redekopp
- All graphics from Wikimedia Commons unless otherwise noted